PATENT
Docket No. **BORL/0217.00**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Jonas Hogstrom et al.	Examiner: Chen, Qing
Serial No.: 10/708,021	Art Unit: 2191
Filed: February 3, 2004	APPEAL BRIEF
For: Development System with Methodology for Run-time Restoration of UML Model from Program Code	

Mail Stop Appeal
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

BRIEF ON BEHALF OF JONAS HOGSTROM

This is an appeal from the Final Rejection mailed June 15, 2007, in which currently-pending claims 1-63 stand finally rejected. Appellant filed a Notice of Appeal on September 19, 2007. This brief is submitted electronically in support of Appellant's appeal.

TABLE OF CONTENTS

1. REAL PARTY IN INTEREST

The real party in interest is assignee Borland Software Corporation, a Delaware corporation, located and doing business at 100 Enterprise Way, Scotts Valley, CA.

2. RELATED APPEALS AND INTERFERENCES

There are no appeals or interferences known to Appellant, the Appellant's legal representative, or assignee which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

3. STATUS OF CLAIMS

The status of all claims in the proceeding is as follows:

Rejected: Claims 1-63

Allowed or Confirmed: None

Withdrawn: None

Objected to: None

Canceled: None

Identification of claims that are being appealed: Claims 1-63

An appendix setting forth the claims involved in the appeal is included as Section 8 of this brief.

4. STATUS OF AMENDMENTS

Two Amendments have been filed in this case. Appellant filed an Amendment on May 25, 2007 in response to a non-final Office Action mailed on January 26, 2007. In the Amendment filed on May 25, 2007, the pending claims were amended in a manner which Appellant believes clearly distinguished the claimed invention over the art of record, for overcoming the art rejections. In response to the Examiner's Final Rejection dated June 15, 2007 (hereinafter "Final Rejection") finally rejecting Appellant's claims, Appellant filed a Notice of Appeal. One Amendment was also entered in this case after the Final Rejection. Appellant filed an Amendment After Final on September 14, 2007 requesting the Examiner to enter amendments to the claims to address non-art objections to claims 2-20, 22, 24-42 and 63 as well as the Examiner's rejection of claim 46 under 35

U.S.C. Section 112, second paragraph and the rejection of claims 23-42 under 35 U.S.C. Section 101. In response to the Amendment After Final, the Examiner issued an Advisory Action with a mailing date of October 31, 2007 which indicated that Appellant's amendments to the claims would be entered and would sufficiently overcome the claim objections as well as the Section 112 and Section 101 rejections. However, the prior-art rejection of Appellant's claims 1-63 was maintained.

5. SUMMARY OF CLAIMED SUBJECT MATTER

As to Appellant's **First Ground** for appeal, Appellant asserts that the art rejection of Appellant's claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 under 35 USC **Section 102(b)** relying on U.S. Patent No. 6,199,195 to Goodwin et al ("Goodwin") fails to teach or suggest all of the claim limitations of Appellant's rejected claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 where the claimed invention is set forth in the embodiment in **independent claim 1:** In a computer system, an improved method for developing and executing an application (Appellant's specification paragraph [0016], paragraphs [0061]-[0062], paragraph [0168]; see generally Figs. 5A-C), the method comprising: creating a model describing business objects and rules of the application (Appellant's specification paragraph [0016], paragraph [0059], paragraph [0061], paragraph [0068], paragraph [0168]; Fig. 3A at 300), creating source code for the application, including representing the model within the source code itself (Appellant's specification paragraph [0016], paragraphs [0060]-[0062], paragraphs [0069]-[0081], paragraph [0168]; Figs. 3B-3E; Fig. 5A at 501), compiling the source code into an executable application (Appellant's specification paragraph [0016], paragraph [0061], paragraph [0168]; Fig. 4 at 420, Fig. 5A at 502), running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application (Appellant's specification paragraph [0016], paragraph [0061], paragraph [0163], paragraph [0168]; Fig. 5A at 503; see generally paragraphs [0051]-[0053] regarding run-time framework), while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework (Appellant's specification paragraph [0016], paragraphs [0061]-[0062], paragraph [0163], paragraphs [0168]-[0175]; Fig. 4 at 450, Fig. 5A at 504-505, Fig. 5B at 511-513, Fig. 5C at 521-522; see generally paragraphs [0148]-[0161] regarding reconstructing model from executable code).

For Appellant's argument under the **First Ground** for appeal, Appellant additionally argues that the art rejection under **Section 102(b)** relying on Goodwin fails to teach or suggest all of the claim limitations of Appellant's rejected claims, where the claimed invention is set forth in the embodiment in **independent claim 23:** In a computer system, an improved system for developing and executing an application (Appellant's specification paragraph [0017], paragraphs [0061]-[0062], paragraphs [0162]-[0163], paragraph [0168]; Fig. 4 at 400; also see generally Figs. 5A-C), the system comprising: a computer system having a processor and memory Appellant's specification paragraph [0017], paragraphs [0039]-[0042], paragraph [0046], paragraph [0165], paragraph [0168]; Fig. 1 at 100), a modeling tool for creating a model describing business objects and rules of the application (Appellant's specification paragraph [0017], paragraph [0059], paragraph [0061], paragraph [0068], paragraph [0168]; Fig. 3A at 300), a module for creating source code for the application and representing the model within the source code itself (Appellant's specification paragraph [0017], paragraphs [0060]-[0062], paragraphs [0069]-[0081], paragraph [0168]; Figs. 3B-3E; Fig. 5A at 501); a compiler for compiling the source code into an executable application Appellant's specification paragraph [0017], paragraph [0061], paragraph [0168]; Fig. 4 at 420, Fig. 5A at 502), and a run-time framework that is able to reconstruct the model from the executable application and use it for providing services (Appellant's specification paragraph [0017], paragraphs [0061]-[0062], paragraph [0163], paragraphs [0168]-[0175]; Fig. 4 at 450, Fig. 5A at 504-505, Fig. 5B at 511-513, Fig. 5C at 521-522; see generally paragraphs [0148]-[0161] regarding reconstructing model from executable code).

For Appellant's argument under the **First Ground** for appeal, Appellant additionally argues that the art rejection under **Section 102(b)** relying on Goodwin fails to teach or suggest all of the claim limitations of Appellant's rejected claims, where the claimed invention is set forth in the embodiment in **independent claim 43:** A method for developing and executing an application on a computer system (Appellant's specification paragraph [0018], paragraphs [0061]-[0062], paragraph [0168]; see generally Figs. 5A-

C), the method comprising: creating a model for developing an application using Unified Modeling Language (UML) technique (Appellant's specification paragraph [0018], paragraph [0059], paragraph [0061], paragraph [0068], paragraph [0168]; Fig. 3A at 300), generating source code to implement the model (Appellant's specification paragraph [0018], paragraphs [0060]-[0062], paragraph [0064], paragraph [0068], paragraph [0168]; Fig. 3B; Fig. 5A at 501), amending the source code for storing model information in the source code (Appellant's specification paragraph [0018], paragraphs [0060]-[0062], paragraphs [0069]-[0081], paragraph [0168]; Figs. 3C-3E; Fig. 5A at 501), compiling the amended source code into an executable application (Appellant's specification paragraph [0018], paragraph [0061], paragraph [0168]; Fig. 4 at 420, Fig. 5A at 502), and running the executable application on the computer system (Appellant's specification paragraph [0018], paragraph [0061], paragraph [0168]; Fig. 5A at 503), reconstructing the model from the executable application (Appellant's specification paragraph [0018], paragraph [0061], paragraph [0163], paragraphs [0168]-[0174]; Fig. 4 at 450, Fig. 5A at 504, Fig. 5B at 511-513; see generally paragraphs [0148]-[0161] regarding reconstructing model from executable code), and making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display (Appellant's specification paragraph [0018], paragraph [0062], paragraph [0163], paragraph [0175]; Fig. 4 at 450, Fig. 5A at 505, Fig. 5C at 521-522).

For Appellant's argument under the **First Ground** for appeal, Appellant additionally argues based on **dependent claims 12, 34 and 53** which includes the claim limitations: after reconstructing the model at run-time, testing integrity of the reconstructed model (Appellant's specification, paragraph [0172]; Fig. 5B at 513).

As to Appellant's **Second Ground** for appeal, Appellant asserts that the art rejection under **Section 103(a)** relying on the combination of Goodwin (above) and U.S. Patent No. 7,103,600 to Mullins ("Mullins") fails to teach or suggest all of the claim limitations of Appellant's rejected claims 7, 8, 29, 30, 48 and 49, where the claimed invention is set forth in the embodiment in **base independent claims 1, 23 and 43** (the mapping of which is shown above under Appellant's **First Ground** for appeal, and which hereby is incorporated by reference). For Appellant's argument under the **Second**

Ground for appeal, Appellant additionally argues based on **dependent claims** 7, **8, 29, 30, 48 and 49** which includes the limitations: using reflection, reading metadata associated with the executable application to create a graph of code elements (Appellant's specification paragraphs [0168]-[0169]; Fig. 5A at 504), spanning the graph for re-creating the model based on code elements encountered (Appellant's specification paragraphs [0168]-[0169]; Fig. 5A at 504, Fig. 5B at 511-513; see also paragraphs [0148]-[0161] and paragraphs [176]-[191] regarding reconstructing model from executable code), and as each code element is encountered, reconstructing a corresponding portion of the model (Appellant's specification paragraph [0166], paragraphs [0168]-[0169]; Fig. 5A at 504, Fig. 5B at 511-513; see also paragraphs [0148]-[0161] paragraphs [176]-[191] regarding reconstructing model from executable code).

As to Appellant's **Third Ground** for appeal, Appellant asserts that the art rejection under **Section 103(a)** relying on the combination of Goodwin (above), Mullin (above) and U.S. Pub. Application 2004/0044990 of Schloegel et al ("Schloegel") fails to teach or suggest all of the claim limitations of Appellant's rejected claims 9, 31 and 50, where the claimed invention is set forth in the embodiment in **base independent claims 1, 23 and 43** (the mapping of which is shown above under Appellant's **First Ground** for appeal, and which hereby is incorporated by reference). For Appellant's argument under the **Third Ground** for appeal, Appellant additionally argues based on **dependent claims 9, 31 and 50** which includes the limitations: traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques (Appellant's specification paragraph [0169]; Fig. 5B at 511).

As to Appellant's **Fourth Ground** for appeal, Appellant asserts that the art rejection under **Section 103(a)** relying on the combination of Goodwin (above) and U.S. Patent No. 6,711,734 issued to Baisley ("Baisley") fails to teach or suggest all of the claim limitations of Appellant's rejected claims 10, 11, 32, 33, 51 and 52, where the claimed invention is set forth in the embodiment in **base independent claims 1, 23 and 43** (the mapping of which is shown above under Appellant's **First Ground** for appeal, and which hereby is incorporated by reference). For Appellant's argument under the **Fourth Ground** for appeal, Appellant additionally argues based on **dependent claims**

10, 11, 32, 33, 51 and 52 which includes the limitations: detecting a class having a package element and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model (Appellant's specification paragraphs [0154]-[0157], paragraph [0166], paragraphs [0169]-[0170]; Fig. 5B at 512), detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package (Appellant's specification paragraphs [0154]-[0157], paragraphs [0169]-[0171]; Fig. 5B at 512).

As to Appellant's **Fifth Ground** for appeal, Appellant asserts that the art rejection under **Section 103(a)** relying on the combination of Goodwin (above) and U.S. Patent No. 7,162,462 issued to Mutschler, III ("Mutschler") fails to teach or suggest all of the claim limitations of Appellant's rejected claims 13, 14, 35, 36, 54 and 55, where the claimed invention is set forth in the embodiment in **base independent claims 1, 23 and 43** (the mapping of which is shown above under Appellant's **First Ground** for appeal, and which hereby is incorporated by reference). For Appellant's argument under the **Fifth Ground** for appeal, Appellant additionally argues based on **dependent claims 13, 14, 35, 36, 54 and 55** which includes the limitations: ensuring that all classes in the model belong to a common superclass (Appellant's specification paragraphs [0172]-[0173]; Fig. 5B at 513), and if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes (Appellant's specification paragraph [0174]; Fig. 5B at 513).

6. GROUNDS OF REJECTION TO BE REVIEWED

The grounds for appeal are:

(1st) Whether claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 are unpatentable under 35 U.S.C. 102(b) as being anticipated by U.S. Patent No. 6,199,195 to Goodwin et al (hereinafter "Goodwin").

(2nd) Whether claims 7, 8, 29, 30, 48 and 49 are unpatentable under 35 U.S.C. 103(a) as being obvious over Goodwin (above) in view of U.S. Patent No. 7,103,600 to Mullins (hereinafter "Mullins").

(3rd) Whether claims 9, 31 and 50 are unpatentable under 35 U.S.C. 103(a) as being obvious over Goodwin (above) in view of Mullins (above), further in view of U.S.

Pub. Application 2004/0044990 of Schloegel et al (hereinafter "Schloegel").

(4th) Whether claims 10, 11, 32, 33, 51 and 52 are unpatentable under 35 U.S.C. 103(a) as being obvious over Goodwin (above) in view of U.S. Patent No. 6,711,734 to Baisley (hereinafter "Baisley").

(5th) Whether claims 13, 14, 35, 36, 54 and 55 are unpatentable under 35 U.S.C. 103(a) as being obvious over Goodwin (above) in view of U.S. Patent No. 7,162,462 to Mutschler, III (hereinafter "Mutschler").

7. ARGUMENT

A. First Ground: Claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 rejected under 35 U.S.C. 102(b)

1. General

Under Section 102, a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in the single prior art reference. (See, e.g., MPEP Section 2131.) As will be shown below, Goodwin fails to teach each and every element set forth in Appellant's claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 (as well as other claims), and therefore fails to establish anticipation of the claimed invention under Section 102.

2. Claims 1-6, 15-28, 37-47, and 56-63

The Examiner has rejected claims 1-6, 12, 15-28, 34, 37-47, 53, and 56-63 under 35 U.S.C. 102(b) as being anticipated by Goodwin. The following rejection of Appellant's claim 1 by the Examiner is representative of the Examiner's rejection of Appellant's claims as being unpatentable over Goodwin:

> As per Claim 1, Goodwin et al. disclose:
>
> - creating a model describing business objects and rules of the application (*see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;
>
> - creating source code for the application, including representing the model within

the source code itself (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..."*);

- compiling the source code into an executable application (*see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JA VA, for the generation of CORBA ORB services ..."*);

- running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application (*see Column 15: 45-65, "The data server 332 operates at run time ... "and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services."*); and

- while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework (*see Column 13: 65-67 through Column 14: 1-9, "Users own code can be combined with the generated code to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for interacting with the particular data model represented in the unified model."; Column 16: 36- 64, "The unified models contain mappings about the data sources of each of the classes. " and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions." and "The data server 332 employs the unified models to provide the Clients 338 periodically updated query-based views of distributed heterogeneous databases."*).

(Final Rejection, paragraph 15, pages 5-6)

At the outset, Appellant does not claim to have invented the notion of using a model for generating application source code; Appellant acknowledges that using a model for generating source code is taught by Goodwin as well as other prior art references. Appellant's invention, however, does not simply generate application code based on a model, but rather <u>Appellant's invention represents the model (e.g., UML model) inside the source code of the application in a manner that enables the model to be reconstructed from the executable application code at runtime</u>. Although Goodwin uses a model to generate code, Goodwin's system does <u>not</u> reconstruct the model from the compiled code of the application, nor does it perform the prior steps necessary to do so -- namely incorporating all necessary components of the model into the source code of the application so that it can be reconstructed from the application at runtime. Instead, <u>Goodwin's solution provides for storing unified models in a schema repository accessed</u>

by a schema server (Goodwin, col. 8, lines 58-62; Fig 3 at 314, 316). The schema repository of Goodwin's system is a SQL-compliant database (Goodwin, col. 10, lines 38-41). Thus, Goodwin relies on persisting a design-time representation of the model in a database. This is not comparable to Appellant's claimed invention.

Appellant's invention provides for representing the original model (e.g., UML model) inside source code generated from the model. Appellant's invention ensures that everything in the model is represented in the emitted source code, so that it can later be reconstructed from the code to a model again (Appellant's specification, paragraphs [0060] - [0061]). This includes not only representing portions of the UML model having a natural representation in the source code (e.g., class names), but also involves adding attributes and other information to the code for conveying meta information about the UML model which is not otherwise expressible in the code (Appellant's specification, paragraph [0070]; Fig. 3D). As an example, consider an association relationship such as a residency association linking Person and Building. When it comes to associations such as this residency association, a standard code representation of such association only captures a fragment of what is represented in a UML model with respect to the association element. With Appellant's invention, however, attributes are used to express important information about the model (e.g., an association element of the model) that cannot be expressed in normal structural coding elements (Appellant's specification, paragraphs [0071] - [0081]). Appellant's solution provides for creating a representation of the model in the source code so that all information from the model can be expressed in the code and subsequently made available at runtime (Appellant's specification, paragraph [0081]). Thus, Appellant's claimed invention incorporates a full representation of the model into the source code itself. This enables the entire model that was used for generating the code to be fully reconstructed back into model form at runtime from the executable code itself as hereinafter described in more detail. These distinctive features are specifically described in Appellant's claims. For example, Appellant's claim 1 includes the following claim limitations:

> In a computer system, an improved method for developing and executing an application, the method comprising:
> creating a model describing business objects and rules of the application;

> creating source code for the application, including representing the model within the source code itself;
>
> (Appellant's claim 1, emphasis added)

Although Goodwin describes generating code from a model, Appellant's review of Goodwin finds no teaching of representing the model within the source code itself as provided in Appellant's specification and claims. The Examiner references Goodwin at column 13, lines 52-55 as including these teachings (Final Rejection, page 44, paragraph 22(b) "Examiner's response"). However, the referenced portion of Goodwin reads as follows:

> Thus, the code generator 330 can support the creation of, for example, IDL, Java or C++ files (CORBA, JAVA RMI and/or COM) based on certain user preferences and selections.
>
> (Goodwin, column 13, lines 52-55).

As illustrated above, the teachings of Goodwin referenced by the Examiner make no mention whatsoever of representing the model inside the source code itself. The Examiner goes on to state that "the generated code is a representation of the model based on the information stored in the model" (Final Rejection, "Examiner's response", paragraph 22(b), page 44). The source code may, in fact, represent some elements of the model which have a natural representation in the source code (e.g., class names). However, Goodwin's conventional steps of generating code based on a model does not ensure that sufficient information from the model is expressed in the code so as to enable the model to be reconstructed from the code at runtime as described in Appellant's specification and claims. This is also evident by the fact that Goodwin makes no mention of reconstructing the model from the executable application at runtime. Instead, Goodwin provides instead for storing unified models in a schema repository which is accessed and used at development time to generate code (Goodwin, column 10, lines 13-18). When access to the model information is needed at runtime, Goodwin's solution consults a persisted development-time representation of the model in the schema repository instead of reconstructing the model from the application code itself as indicated by the following:

> The models are retrieved through the schema server 316 by, for example, the code generator 330 (at development time) and the data server 332 (at run time). Various adaptors are used by the repository adaptor tool 312 to implement a common interface between the various supported modeling tools and the schema server 316. The unified models are stored in the schema repository 314. The repository adaptor tool 312 initially connects to the schema server 316 to see if a particular unified model for a data source that a user wants to use already exists. It also connects to commit a unified model for storage in the schema repository 314.
>
> (Goodwin, column 10, line 63 - column 11, line 6, emphasis added)

In contrast to Goodwin' which relies on storing unified model information in a database, Appellant's invention provides storing the model information inside the executable application code. This enables the model to be reconstructed at runtime from the executable code of the application (Appellant's specification, paragraph [0061]). This feature of reconstructing the model from the executable code at runtime is only possible because of the prior step of incorporating a representation of the model in the source code. As Goodwin's solution does not represent the model in the code, it also cannot reconstruct the model from the code as doing so requires that the necessary representation of the model is included in the code in the first place.

With Appellant's solution, one need not have the original UML model available (e.g., persistently stored in a repository) at the point the model is reconstructed at runtime. Instead, Appellant's invention enables the original model to be reconstructed at runtime from the executable application (Appellant's specification, paragraphs [0060]-[0061] and paragraphs [0168]-[0169]; see also Fig. 5A at 504-505 and Fig. 5B at 511-513). This feature of reconstructing the model from the executable application is specifically indicated in Appellant's claims, including, for example, the following limitations of Appellant's claim 1:

> compiling the source code into an executable application;
> running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application; and
> while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework.
>
> (Appellant's claim 1, emphasis added)

Goodwin has no comparable teaching of reconstructing the model from the executable application at run time. At paragraph 22(c) on page 45 of the Final Rejection, the Examiner references two different sections of Goodwin as including equivalent teachings. However, review of the referenced teachings finds that they do not describe anything at all analogous to Appellant's claimed invention.

The Examiner first references Goodwin at column 13, lines 65-67 through column 14, lines 1-9 as including teachings of reconstructing the model from the executable application and making it available to the run-time framework. However, the referenced portion of Goodwin provides as follows:

> Output from the code generator 330 can be combined with other user defined codes to create an application. The code generator 330 reads in unified models from the schema server 330 or from an Interface Definition Language (IDL) file and then applies the set of templates specified in the properties file to support each of the selected services. User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model.
>
> (Goodwin, column 13, lines 65-67 through column 14, lines 1-9)

The above simply describes generating code from a unified model which is read in from a schema server. No mention is made of reconstructing the model from the executable application.

The Examiner also references Goodwin at column 16: lines 36-64 for the teachings of reconstructing the model from the executable application. Here, Goodwin is describing features of a data server component which is also depicted as data server 332 at Fig. 3 of Goodwin. Goodwin's data server 332 accepts queries from clients (e.g., client application 338 as shown at Fig. 3) in standard formats which it then interacts with legacy data sources (e.g., schema server 316) to obtain the requested information (See also, Goodwin, column 16, lines 56-59). In other words this data server provides a uniform, object-oriented access to distributed legacy data sources by acting as an object oriented dynamic front into existing databases (Goodwin, column 16, lines 59-61; see also Goodwin, col. 10, lines 63-65). Thus, Goodwin's system relies on retrieving model information stored in a data repository and does not reconstruct the model from the executable application. This clearly shown at Fig. 3 of Goodwin which depicts the client application 338 requesting information from the schema server 316 -- an external source. Also, the schema server itself doesn't reconstruct any model. The schema server is fed at design time with the original unified model that was used to generate code and the model is persisted in a schema repository. Retrieving a stored copy of a model generated at design time from a repository is not equivalent to Appellant's claim limitations of reconstructing the model from the executable application (compiled code).

Advantages of Appellant's claimed invention of storing the model in the compiled code include that the model information and the code are synchronized. With Appellant's approach there is only one representation of the model. With Goodwin's solution, in contrast, there may be a mismatch between the compiled code and the stored model which was generated and persisted in the schema repository. The code and the stored model may, in fact, represent different versions from different points in time. For example, the code may have been modified after the model was created and stored at design time. Alternatively, the model stored in the repository of Goodwin's system may have been updated, but the generated code may not reflect the most recent updates to the model. With Appellant's approach of storing the model in the compiled code, there is only one representation of the model which is synchronized with the code as the model is represented inside the code itself.

3. Claims 12, 34 and 53

Further distinctions between Goodwin and Appellant's solutions can be found in Appellant's dependent claims. For example, Appellants claims 12, 34 and 53 include claim limitations of testing integrity of the reconstructed model after the model is reconstructed at runtime. The Examiner references the following portion of Goodwin as including the equivalent teachings:

> Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model (unified model 206 (Fig. 2)).
>
> (Goodwin, col. 10, lines 7-11)

The above-referenced portion of Goodwin simply describes a interface to the model meta-information through the schema server. It is not at all analogous to Appellant's claimed limitations of testing integrity of the model after it is reconstructed.

4. Conclusion

All told, Goodwin provides no teaching of representing the model inside the source code so as to enable reconstructing the model from the executable code at runtime as described in Appellant's specification and claims. Therefore, Goodwin does not anticipate Appellant's invention as it does not teach each and every element set forth in Appellant's claims 1-6, 12, 15-28, 34, 37, 39-47, 53, and 56-63 (and other claims). For the reasons stated, it is respectfully submitted that Appellant's claims 1-6, 12, 15-28, 34, 37, 39-47, 53, and 56-63 distinguish over Goodwin and that the Examiner's rejection of these claims under Section 103 should not be sustained.

B. Second Ground: Claims 7, 8, 29, 30, 48 and 49 rejected under 35 U.S.C. 103(a)

1. General

Under Section 103(a), a patent may not be obtained if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. To establish a prima facie case of obviousness under this section, the Examiner must establish: (1) that there is some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings, (2) that there is a reasonable expectation of success, and (3) that the prior art reference (or references when combined) must teach or suggest all the claim limitations. (See e.g., MPEP 2142). The reference(s) cited by the Examiner fail to meet these conditions.

2. Claims 7, 8, 29, 30, 48 and 49

The Examiner has rejected claims 7, 8, 29, 30, 48 and 49 under 35 U.S.C. 103(a) as being obvious over Goodwin in view of U.S. Patent No. 7,103,600 to Mullins ("Mullins"). As to these claims, the Examiner continues to rely on Goodwin as substantially teaching Appellant's invention, but acknowledges that Goodwin does not teach "using reflection, reading metadata associated with the executable application to create a graph of code elements; and spanning the graph for recreating the model based on code elements encountered" (Final Rejection, paragraph 17, page 22). The Examiner therefore adds Mullins for these teachings.

Claims 7, 8, 29, 30, 48 and 49 are dependent upon Appellant's independent claims 1, 23 and 43 and, therefore, are believed to be allowable for at least the reasons cited above under Appellant's First Ground of appeal (incorporated by reference herein) pertaining to the deficiencies of Goodwin in respect to Appellant's invention. In particular, Goodwin does not include teachings of representing the model inside the source code so as to enable reconstructing the model from the executable code at runtime as described in Appellant's specification and claims. Claims 7, 8, 29, 30, 48 and 49 are also believed to be patentable for the following additional reasons.

Mullins discusses use of reflection in a system for creating and maintaining distributed transparent persistence of data objects and data object graphs (Mullins, Abstract). However, it does not cure any of the above-described deficiencies of Goodwin as it provides no comparable teaching of representing the model within application code or of reconstructing the model from the executable application at runtime. The teachings of Mullins referenced by the Examiner describe use of runtime classes to persist instances of a compiled application for use in developing a navigation model for the application (Mullins, column 36 lines 50-67). Although Mullins describes creation of a navigation model, Appellant respectfully fails to see how this is analogous to Appellant's invention for reconstructing a unified model from executable code as provided, for example in the following limitations of Appellant's claim 7:

> The improved method of claim 1, wherein the reconstructing step includes:
> using reflection, reading metadata associated with the executable application to create a graph of code elements; and
> spanning the graph for re-creating the model based on code elements encountered.
>
> (Appellant's claim 7)

For one thing, Mullin's system does not re-construct or re-create a model from executable application code. Instead, Mullin's solution provides for persisting or storing a "CDOG model" in a repository such as a relational database. As described at column

35, lines 4-11, Mullin's system includes a runtime O/R mapping class that wrappers the JDBC driver, issues queries and does the actual persistence calls (e.g., to a relational database). As shown at column 35, lines 50-66, Mullin's solution opens a connection to the database to retrieve data objects. Thus, Mullins provides for persisting complex data object models to a data store or information repository external to the application code (see also, Mullins, col. 11, lines 4-27). In contrast, Appellant's approach involves reading the code modules themselves (Appellant's specification, paragraphs [179]-[180] and paragraphs [148]-[152]). Thus, Appellant's approach reconstructs the model based on information stored inside the executable code itself while Mullins retrieves information stored in an external repository (e.g., database). In addition, Appellant notes that Mullins' system cannot reconstruct the model from the executable application as the necessary steps to represent the model inside the code have not been undertaken in the first place.

3. Conclusion

As described above, Mullins includes no teachings which cure the deficiencies of the Goodwin reference as to Appellant's invention. In addition, Mullins does not appear to includes the specific teachings of creating a graph of code elements from the executable application and spanning the graph for re-creating the model based on code elements encountered as described in Appellant's specification and claims. For the reasons stated, it is respectfully submitted that Appellant's claims distinguish over the prior art and that the Examiner's rejection under Section 103 should not be sustained.

C. Third Ground: Claims 9, 31 and 50 rejected under 35 U.S.C. 103(a)

1. Claims 9, 31 and 50

The Examiner has rejected claims 9, 31 and 50 under 35 U.S.C. 103(a) as being obvious over Goodwin in view of Mullins, further in view of U.S. Pub. Application 2004/0044990 of Schloegel et al ("Schloegel"). As to these claims, the Examiner acknowledges that Goodwin and Mullins do not disclose spanning the graph using a selected one of depth-first, breadth-first and ad-hoc traversal techniques and thus adds Schloegel for these teachings (Final Rejection, page 28, paragraph 18).

Claims 9, 31 and 50 are dependent upon Appellant's independent claims 1, 23 and 43 as well as dependent claims 7, 29 and 48 and, therefore, are believed to be allowable for at least the reasons cited above pertaining to the deficiencies of Goodwin and Mullins in respect to Appellant's invention. As described under Appellant's First Ground of appeal and Second Ground of appeal (both incorporated by reference herein), neither Goodwin nor Mullins teaches representing the model inside the source code so as to enable reconstructing the model from the executable code at runtime as described in Appellant's specification and claims. Schloegel does not cure these deficiencies of Goodwin and Mullins as to Appellant's invention.

Schloegel describes alternative techniques for traversal during code generation as follows:

> The order of entity traversal during code generation can be application specific. For example, <u>the order of entity traversal during code generation</u> could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.
>
> (Schloegel, paragraph [0033], emphasis added).

As shown above, Schloegel discusses traversal during code generation, while Appellant's claimed invention creates a graph of code elements based on the executable application itself and then spans the graph using depth-first, breadth-first or ad-hoc traversal techniques to reconstruct the model from the executable application. As Schloegel does not include any teaching of representing a unified model within application code generated from the model or of reconstructing the model from the executable application at runtime, it does not cure any of the above-described deficiencies of Goodwin and Mullins as to Appellant's invention.

2. Conclusion

As the combined references do not teach or suggest all the claimed features of Appellant's invention, it is respectfully submitted that Appellant's claimed invention as set forth by these claims is distinguishable over the combined references, and that the rejection of claims 9, 31 and 50 under Section 103 should not be sustained.

D. Fourth Ground: Claims 10, 11, 32, 33, 51 and 52 rejected under 35 U.S.C. 103(a)

The Examiner has rejected claims 10, 11, 32, 33, 51 and 52 under 35 U.S.C.

103(a) as being obvious over Goodwin in view of U.S. Patent No. 6,711,734 to Baisley ("Baisley"). The Examiner again relies on Goodwin as substantially teaching the claimed invention, but acknowledges that Goodwin does not disclose creating a UML package for the reconstructed model based on detecting a class having a package element or an attribute specifying that a class belongs to a UML package (Final Rejection, page 31, paragraph 19). Therefore, the Examiner adds Baisley for these teachings.

Claims 10, 11, 32, 33, 51 and 52 are dependent upon Appellant's independent claims 1, 23 and 43 and, therefore, are believed to be allowable for at least the reasons cited above pertaining to the above-described deficiencies of Goodwin as to Appellant's invention. As described under Appellant's First Ground of appeal (incorporated by reference herein), Goodwin does not teach representing the model inside the source code so as to enable reconstructing the model from the executable code at runtime. Baisley does not cure these deficiencies of Goodwin.

Baisley describes automatically translating a stored Metaobject Facility (MOF) metamodel into a Unified Modeling Language (UML) Model (Baisley, Abstract). However, Baisley does not provide any teaching of incorporating the UML model into the source code of the application or of reconstructing a UML model from the executable application as described in Appellant's specification and claims. Moreover, Appellant's review of the Bailey reference indicates that it also does not include the specific teachings of detecting a class having a package element and creating a corresponding UML package for the reconstructed model as provided, for example, in Appellant's claim 10. Instead, Baisley describes determining whether a MOF package which is being translated into a UML model contains "clustered imports" (Baisley, Fig. 3 at 33; column 4, lines 29-36). If the MOF package contains clustered imports, then Baisley's system creates a "clusteredImport" tagged value on the UML model (Baisley, Fig. 3 at 34; column 4, lines 36-39). Appellant does not believe that this is equivalent to Appellant's claim limitations of detecting a class having a package element and creating a corresponding UML package. Among other things, Baisley makes no mention of detecting a class as, unlike Appellant's invention, Baisley is not reconstructing a model from an executable application.

2. Conclusion

The Goodwin and Baisley references, even when combined, do not teach or suggest all the claimed features of Appellant's invention. Therefore, Appellant respectfully submits that Appellant's claimed invention is distinguishable over the combined references, and that the Examiner's rejection of claims 10, 11, 32, 33, 51 and 52 under Section 103 should not be sustained.

E. Fifth Ground: Claims 13, 14, 35, 36, 54 and 55 rejected under 35 U.S.C. 103(a)

The Examiner has rejected claims 13, 14, 35, 36, 54 and 55 under 35 U.S.C. 103(a) as being obvious over Goodwin in view of U.S. Patent No. 7,162,462 to Mutschler, III ("Mutschler"). The Examiner again relies on Goodwin, but acknowledges that Goodwin does not disclose constructing a common superclass for classes of the reconstructed model and, therefore, adds Mutschler for this teaching (Final Rejection, paragraph 20, pages 36-37).

Claims 13, 14, 35, 36, 54 and 55 are dependent upon Appellant's independent claims 1, 23 and 43 as well as dependent claims 12, 34 and 53 and, therefore, are believed to be allowable for at least the reasons cited above under Appellant's First Ground of appeal (incorporated by reference herein) as to the deficiencies of Goodwin with respect to Appellant's invention. Mutschler does not cure any of these deficiencies as it simply describes the creation of a superclass for both events and rules, allowing them to share data and methods relating to their common dynamic behavior over time (Mutschler, col. 5, lines 40-43) in the context of a system providing time sensitivity to an inference engine (Mutschler, Abstract). Appellant's claimed invention, in contrast, ensures that all classes in the model belong to a common superclass which may include constructing a common superclass if it is found that all classes in the reconstructed model do not share a common superclass (Appellant's Fig. 5B at 513; paragraphs [172]-[174]). Appellant's invention performs these steps during the process of traversing the reconstructed model information to ensure integrity of the model (Appellant's Fig. 5B at 513). Neither Goodwin nor Mutschler perform this type of integrity testing of a model by ensuring all classes in the model belong to a common superclass.

2. Conclusion

The Goodwin and Mutschler references, even when combined, do not teach or suggest all the claimed features of Appellant's invention. Accordingly, it is respectfully submitted that Appellant's claimed invention is distinguishable over the combined references, and that the Examiner's rejection of claims 13, 14, 35, 36, 54 and 55 under Section 103 should not be sustained.

C. Conclusion

The present invention greatly improves the efficiency of the task of synchronizing a unified model used in creation of an application with the application itself by storing information about the model in the compiled code itself. Appellant's approach ensures there is only one consistent representation of the model, thereby avoiding a mismatch between the model actually expressed in the code and a persisted version of the model (e.g., model information stored in an external repository). It is respectfully submitted that the present invention, as set forth in the pending claims, sets forth a patentable advance over the art.

In view of the above, it is respectfully submitted that the Examiner's rejection of Appellant's claims under 35 U.S.C. Section 102 and 35 U.S.C. Section 103 should not be sustained. If needed, Appellant's undersigned attorney can be reached at 925 465 0361. For the fee due for this Appeal Brief, please refer to the attached Fee Transmittal Sheet. This Appeal Brief is submitted electronically in support of Appellant's Appeal.

Respectfully submitted,

Date: November 19, 2007

/G. Mack Riddle/

G. Mack Riddle; Reg. No. 55,572
Attorney of Record

925 465 0361
925 465 8143 FAX

8. CLAIMS APPENDIX

1. In a computer system, an improved method for developing and executing an application, the method comprising:

creating a model describing business objects and rules of the application;

creating source code for the application, including representing the model within the source code itself;

compiling the source code into an executable application;

running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application; and

while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework.

2. The improved method of claim 1, wherein the model comprises a Unified Modeling Language (UML) model.

3. The improved method of claim 1, wherein the source code is created using a programming language.

4. The improved method of claim 3, wherein the programming language is an object oriented programming language.

5. The improved method of claim 3, wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application.

6. The improved method of claim 1, wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application.

7. The improved method of claim 1, wherein the reconstructing step includes:

using reflection, reading metadata associated with the executable application to create a graph of code elements; and

spanning the graph for re-creating the model based on code elements encountered.

8. The improved method of claim 7, wherein the spanning step includes:

as each code element is encountered, reconstructing a corresponding portion of the model.

9. The improved method of claim 7, wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

10. The improved method of claim 1, wherein the reconstructing step includes:

detecting a class having a package element; and

creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

11. The improved method of claim 10, further comprising:

detecting an attribute specifying that a class belongs to the UML package; and

specifying in the reconstructed model that the class belongs to that UML package.

12. The improved method of claim 1, further comprising:

after reconstructing the model at run-time, testing integrity of the reconstructed model.

13. The improved method of claim 12, further comprising:

ensuring that all classes in the model belong to a common superclass.

14. The improved method of claim 13, further comprising:

if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

15. The improved method of claim 1, wherein the reconstructed model is stored in a cache memory available to the run-time framework.

16. The improved method of claim 1, wherein the model is initially created using a modeling tool, and wherein the source code is compiled using a compiler.

17. The improved method of claim 1, wherein the step of creating source code includes:

representing information of the model in source code as language constructs.

18. (The improved method of claim 1, wherein the step of creating source code includes:

representing information of the model in source code as attributes.

19. The improved method of claim 18, wherein attributes comprise specifiers to structural code elements.

20. The improved method of claim 1, wherein the step of creating source code includes:

representing information of the model in code artifacts that exist expressly for carrying model information in source code.

21. A computer-readable medium having processor-executable instructions for performing the improved method of claim 1.

22. A downloadable set of processor-executable instructions for performing the method of claim 1 stored on a computer-readable medium.

23. In a computer system, an improved system for developing and executing an application, the system comprising:

a computer system having a processor and memory;

a modeling tool for creating a model describing business objects and rules of the application;

a module for creating source code for the application and representing the model within the source code itself;

a compiler for compiling the source code into an executable application; and

a run-time framework that is able to reconstruct the model from the executable application and use it for providing services.

24. The improved system of claim 23, wherein the model comprises a Unified Modeling Language (UML) model.

25. The improved system of claim 23, wherein the source code is created using a programming language.

26. The improved system of claim 25, wherein the programming language is an object oriented programming language.

27. The improved system of claim 25, wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application.

28. The improved system of claim 23, wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application.

29. The improved system of claim 23, wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered.

30. The improved system of claim 29, wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application.

31. The improved system of claim 29, wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

32. The improved system of claim 23, wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

33. The improved system of claim 32, further comprising:

a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package.

34. The improved system of claim 23, further comprising:

a submodule for testing integrity of the reconstructed model.

35. The improved system of claim 34, further comprising:

a submodule for ensuring that all classes in the model belong to a common superclass.

36. The improved system of claim 35, further comprising:

a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass.

37. The improved system of claim 23, wherein the reconstructed model is stored in a cache memory available to the run-time framework.

38. The improved system of claim 23, wherein the model is initially created using a UML modeling tool, and wherein the source code is compiled using a C# compiler.

39. The improved system of claim 23, wherein the module for creating source code is able to represent information of the model in source code as language constructs.

40. The improved system of claim 23, wherein the module for creating source code is able to represent information of the model in source code as attributes.

41. The improved system of claim 40, wherein attributes comprise specifiers to structural code elements.

42. The improved system of claim 23, wherein the module for creating source code is able to represent information of the model in code artifacts that exist expressly for carrying model information in source code.

43. A method for developing and executing an application on a computer system, the method comprising:

creating a model for developing an application using Unified Modeling Language (UML) technique;

generating source code to implement the model;

amending the source code for storing model information in the source code;

compiling the amended source code into an executable application and running the executable application on the computer system;

reconstructing the model from the executable application; and

making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display.

44. The method of claim 43, wherein the source code is implemented using a programming language.

45. The method of claim 44, wherein the programming language is an object oriented programming language.

46. The method of claim 45, wherein the object oriented programming language is one that supports reflection technique, thereby allowing reconstruction of the model from the executable application.

47. The method of claim 43, wherein the reconstructed model is employed by a run-time framework to provide services to the executable application.

48. The method of claim 43, wherein the reconstructing step includes:

using reflection, reading metadata associated with the executable application to create a graph of code elements; and

spanning the graph for re-creating the model based on code elements encountered.

49. The method of claim 48, wherein the spanning step includes:

as each code element is encountered, reconstructing a corresponding portion of the model.

50. The method of claim 48, wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

51. The method of claim 43, wherein the reconstructing step includes:

detecting a class having a package element; and

creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

52. The method of claim 51, further comprising:

detecting an attribute specifying that a class belongs to the UML package; and

specifying in the reconstructed model that the class belongs to that UML package.

53. The method of claim 43, further comprising:

after reconstructing the model, testing integrity of the reconstructed model.

54. The method of claim 53, further comprising:

ensuring that all classes in the model belong to a common superclass.

55. The method of claim 54, further comprising:

if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

56. The method of claim 43, wherein the reconstructed model is stored in a cache memory.

57. The method of claim 43, wherein the model is initially created using a modeling tool, and wherein the amended source code is compiled using a compiler.

58. The method of claim 43, wherein the step of amending the source code includes:

representing information of the model in source code as language constructs.

59. The method of claim 43, wherein the step of amending the source code includes:

representing information of the model in source code as attributes.

60. The method of claim 59, wherein attributes comprise specifiers to structural code elements.

61. The method of claim 43, wherein the step of amending the source code includes:

representing information of the model in code artifacts that exist expressly for carrying model information in source code.

62. A computer-readable medium having processor-executable instructions for performing the method of claim 43.

63. A downloadable set of processor-executable instructions for performing the method of claim 43 stored on a computer-readable medium.

9. EVIDENCE APPENDIX

This Appeal Brief is not accompanied by an evidence submission under §§ 1.130, 1.131, or 1.132.

10. RELATED PROCEEDINGS APPENDIX

Pursuant to Appellant's statement under Section 2, this Appeal Brief is not accompanied by any copies of decisions.